

Лабораторная работа №6

Аутентификация и авторизация

1. Цель работы.

Знакомство с механизмом аутентификации и авторизации в ASP.Net Core.

2. Задача работы

Научиться выполнять аутентификацию и авторизацию на удаленном сервере. Научиться ограничивать доступ к API для незарегистрированных пользователей.

Время выполнения работы: 6 часов (3 занятия)

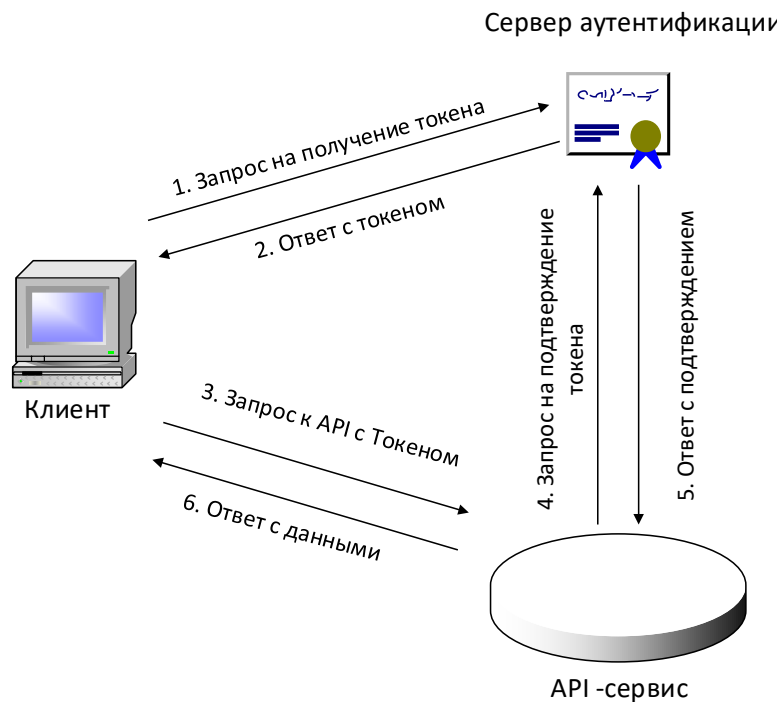
3. Предварительная информация

3.1. Выбор сервера аутентификации

Для более детального ознакомления с аутентификацией и авторизацией в ASP.NET Core в данной работе предлагается использовать отдельное приложение – сервер аутентификации. Данный сервер должен предоставить страницы входа в систему, регистрации в системе, предоставление токена аутентификации и возможность проверки токена.

Поскольку доступ к готовым серверам аутентификации, таким как Azure Active Directory, ограничен в связи с санкциями, предлагается создать свое приложение – сервер аутентификации. В качестве такого сервера воспользуемся готовым решением – Keycloak (<https://www.keycloak.org/>)

Схема взаимодействия с сервером аутентификации при работе с API выглядит так:



3.2. Некоторые термины Keycloak

3.2.1. Realm

В контексте Keycloak Realm представляет собой домен безопасности и администрирования, в котором осуществляется управление пользователями, приложениями и ролями. Это фундаментальная концепция архитектуры Keycloak, которая позволяет изолировать и организовывать ресурсы, разрешения и конфигурации.

3.2.2. API scope

Первоначальная спецификация OAuth 2.0 имеет концепцию областей (scope), которая просто определяется как область доступа, которую запрашивает клиент при работе с API.

Пример регистрации областей:

```
public static IEnumerable<ApiScope> ApiScopes =>
    new ApiScope[]
    {
        new ApiScope("scope1"),
        new ApiScope("scope2"),
    };
```

3.2.3. Clients

В Keycloak Client относится к приложению или сервису, который взаимодействует с сервером Keycloak для целей аутентификации и авторизации. Это может быть веб-приложение, мобильное приложение, серверный API или любой другой тип приложения, которому необходимо аутентифицировать и авторизовать своих пользователей.

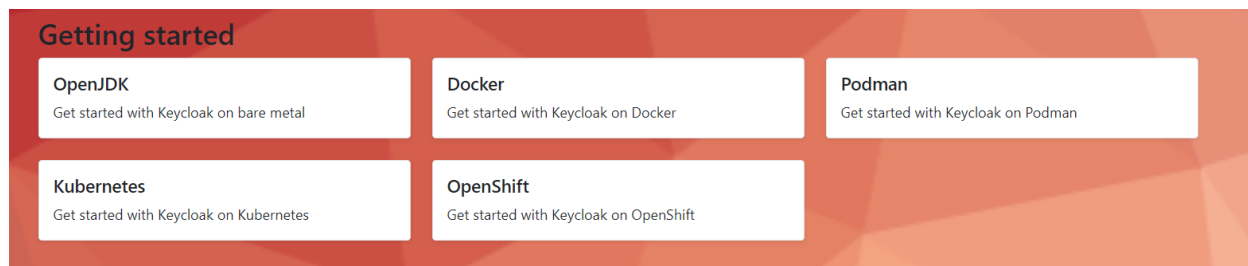
Клиенты создаются внутри Realm.

4. Выполнение работы.

4.1. Установка Keycloak

Документация Keycloak на сайте <https://www.keycloak.org/guides>

Выберите способ запуска сервера Keycloak:



В приведенных примерах будет использоваться OpenJDK (<https://www.keycloak.org/getting-started/getting-started-zip>)

Установка OpenJDK (можно использовать любую другую Java-машину)
- см. <https://learn.microsoft.com/ru-ru/java/openjdk/install>)

Назначьте сертификат SSL

4.2. Подключение базы данных к Keycloak

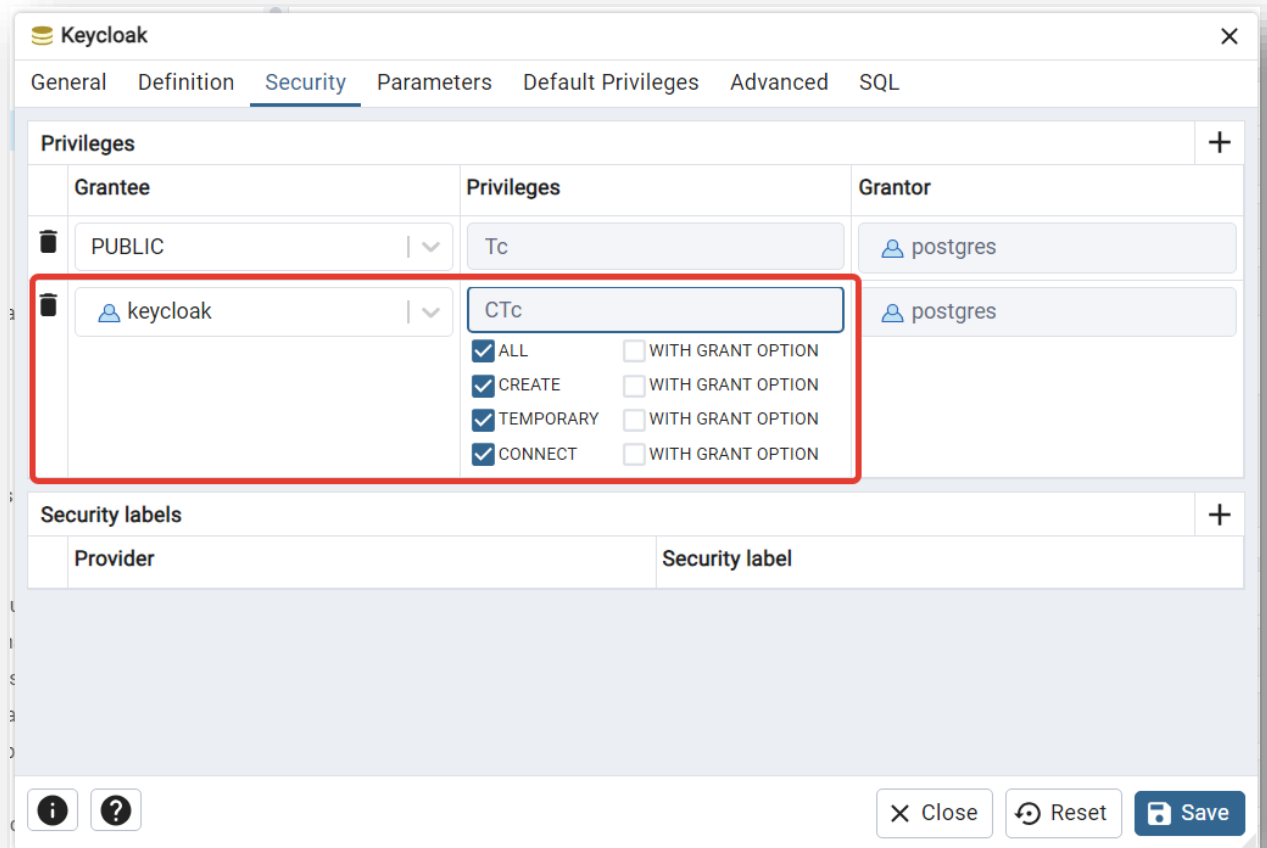
В приведенных примерах используется PostgreSQL. Вы можете использовать любую другую базу данных

Загрузите и установите PostgreSQL (<https://www.postgresql.org/download/>)

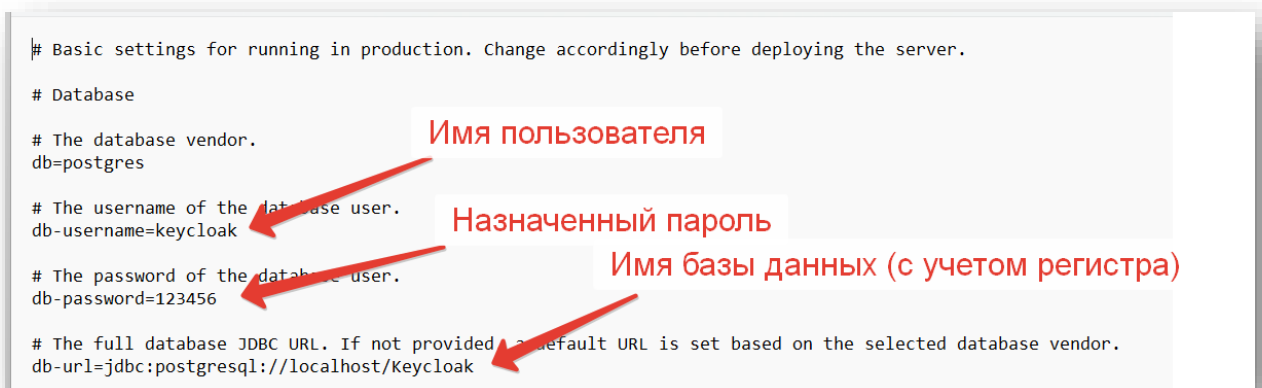
Создайте базу данных (например Keycloak)

Создайте пользователя (например keycloak)

Добавьте созданного пользователя в базу данных и дайте ему все права:



Укажите использование базы данных в keycloak. Для этого откройте файл на риснке, с указанием ваших данных:



4.3. Настройка Keycloak

Для удобства запуска Keycloak можно создать файл xxx.bat (xxx- любое удобное для вас имя) с таким содержимым:

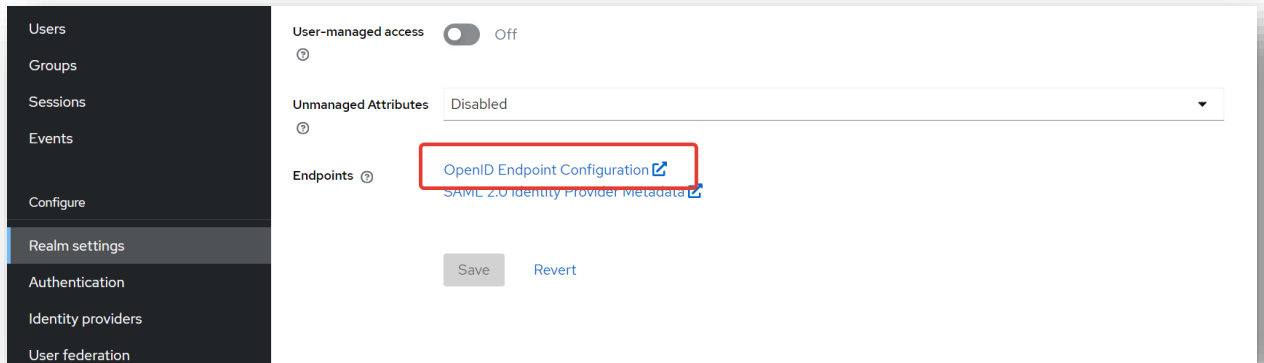
start bin\kc.bat start-dev

Запустите Keycloak.

4.3.1. Создание Realm.

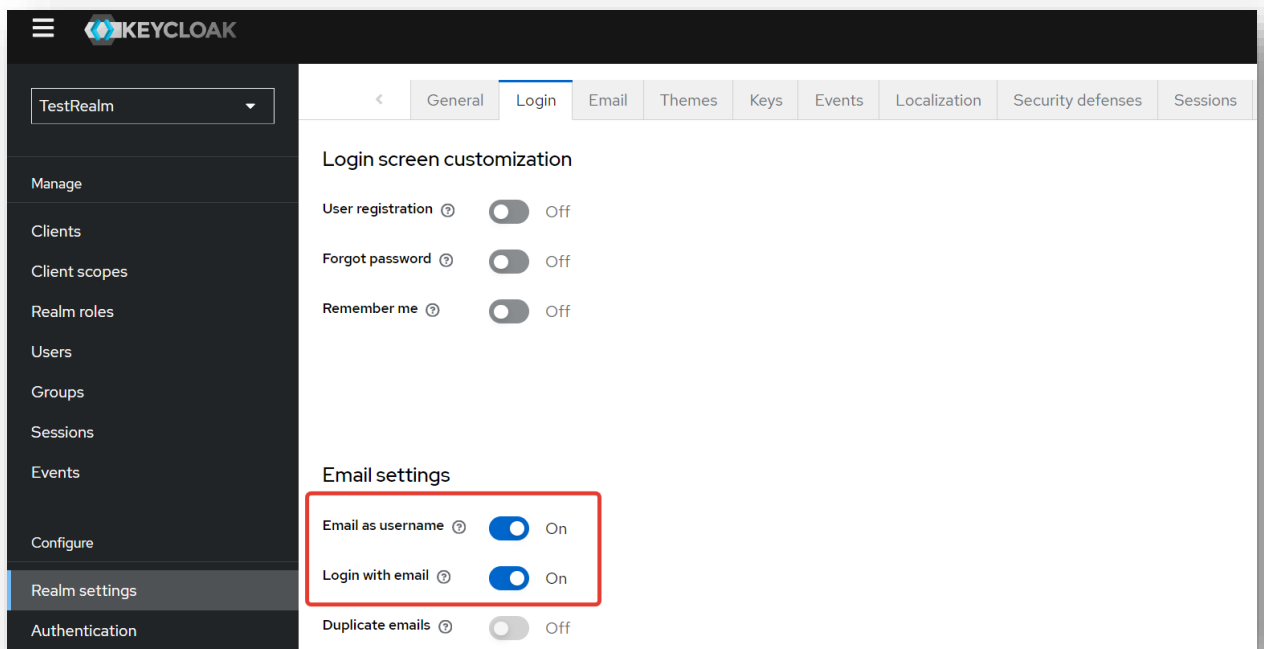
В качестве названия для Realm используйте вашу фамилию

Откройте вкладку Realm Settings. Кликните ссылку конечных точек.



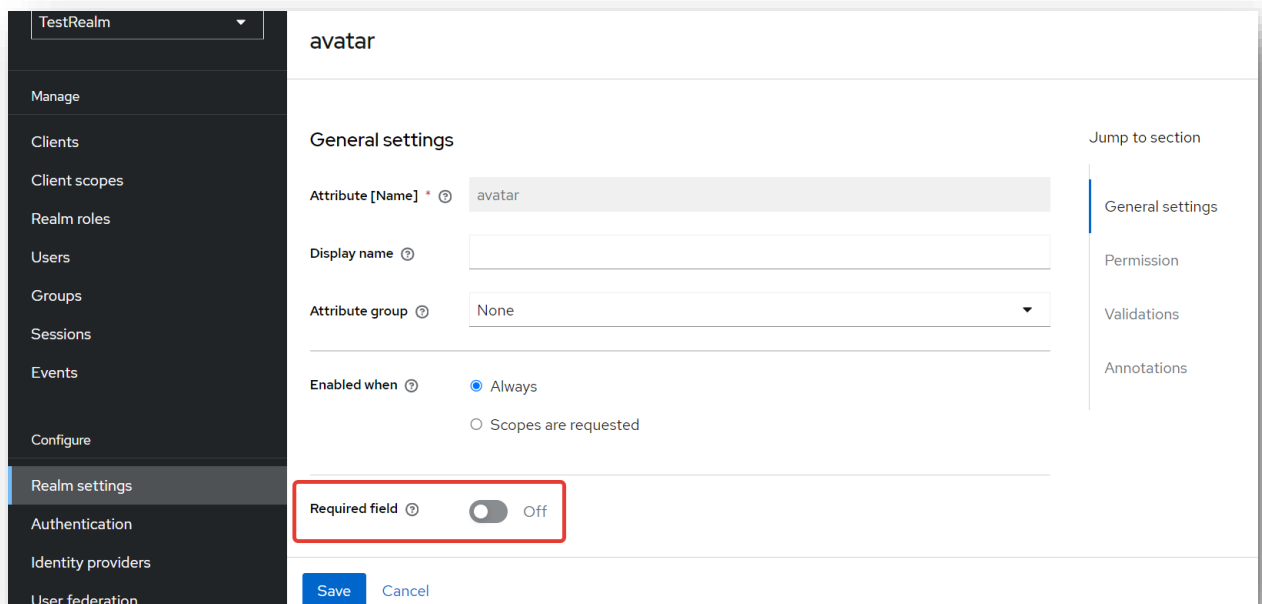
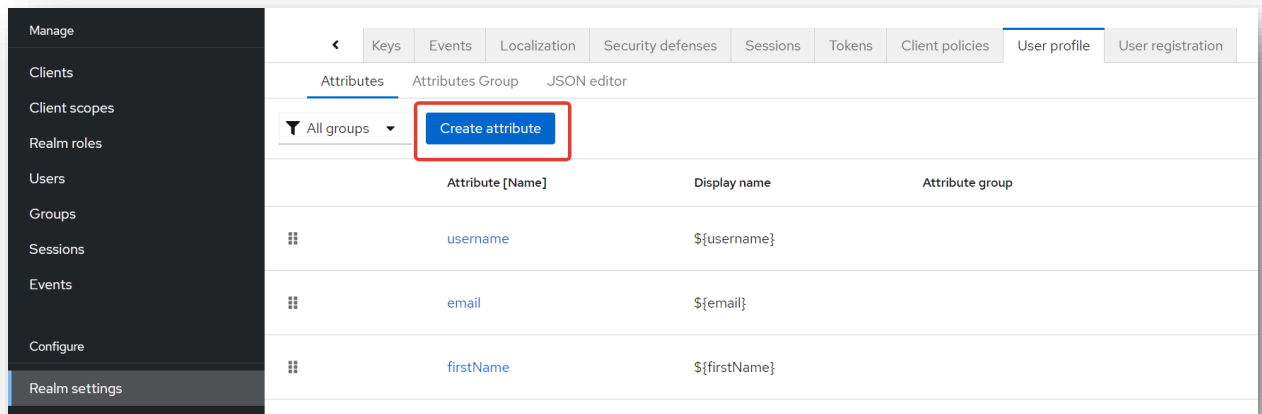
Найдите адреса конечных точек для доступа к вашему Realm.

На вкладке Login установите использование email в качестве имени пользователя:



На вкладке User Profile удалите поля FirstName и LastName (мы не будем их использовать)

На вкладке User Profile добавьте для пользователя необязательный атрибут avatar для хранения Url аватара пользователя.

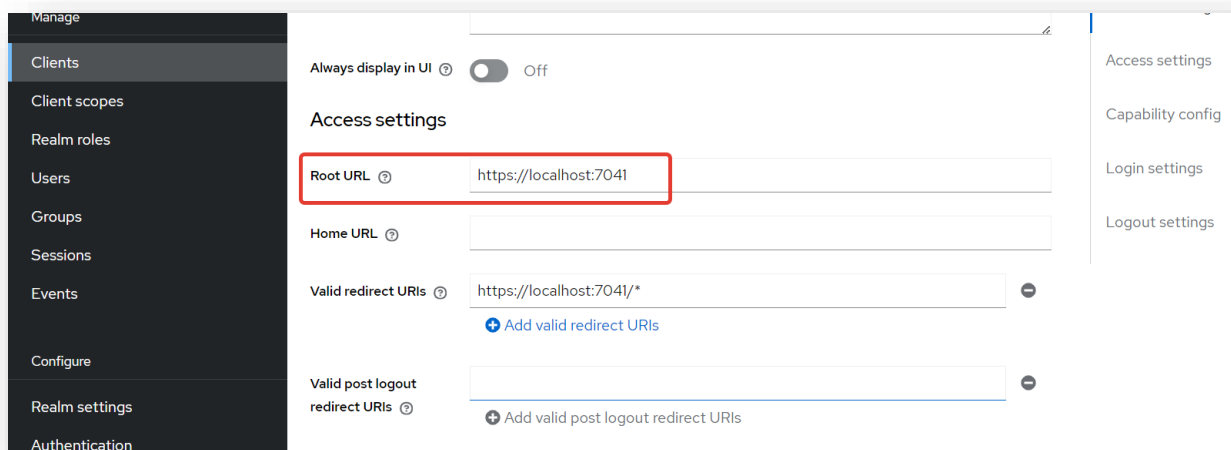


4.3.2. Создание клиента.

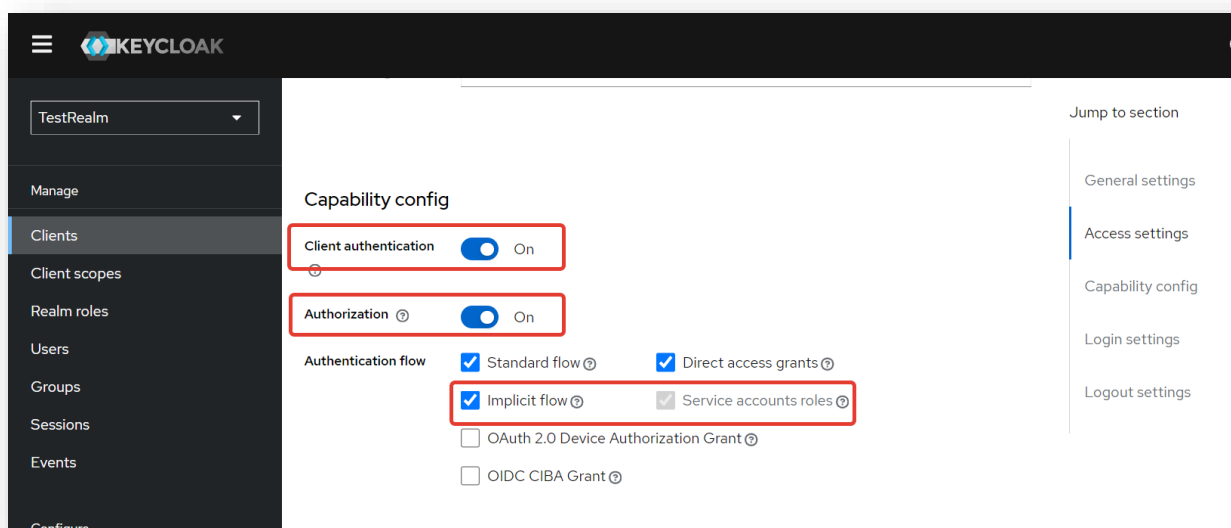
Используйте название [Ваша фамилия]UiClient

Перейдите на вкладку Clients. Выберите созданного клиента

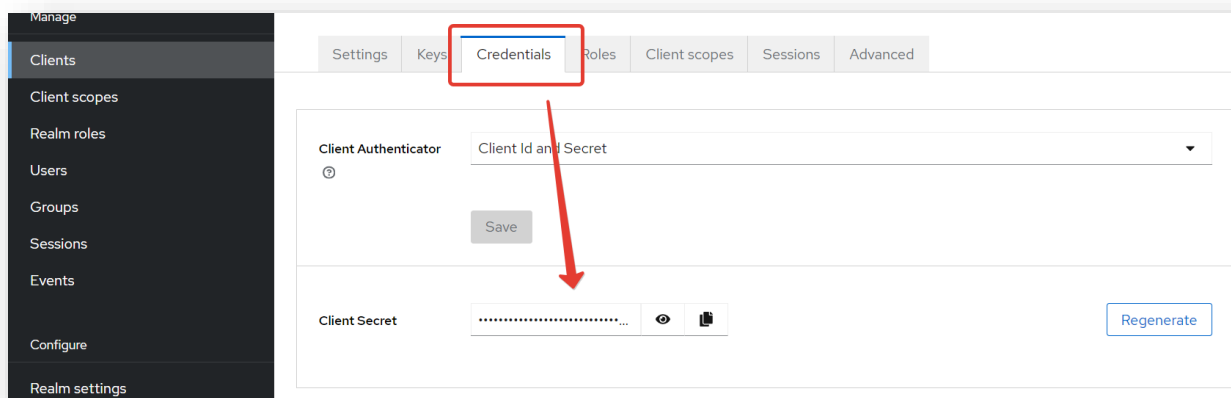
Укажите Url вашего приложения:



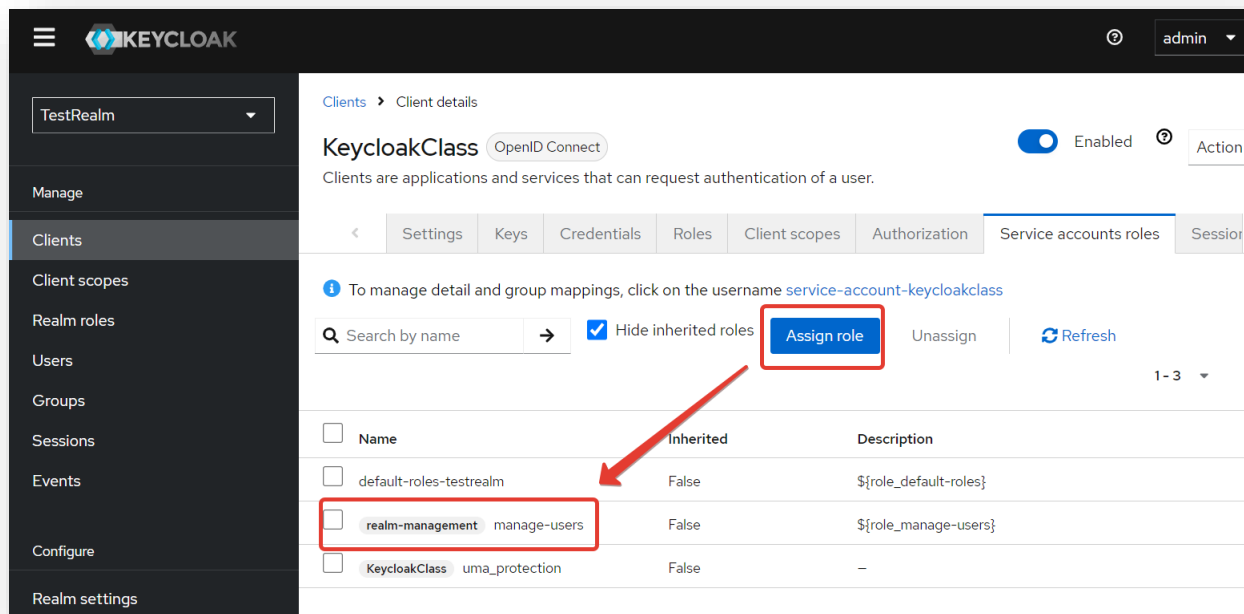
Разрешите Client Authentication, Authorization и Implicit Flow:



На вкладке Credentials найдите поле Client Secret



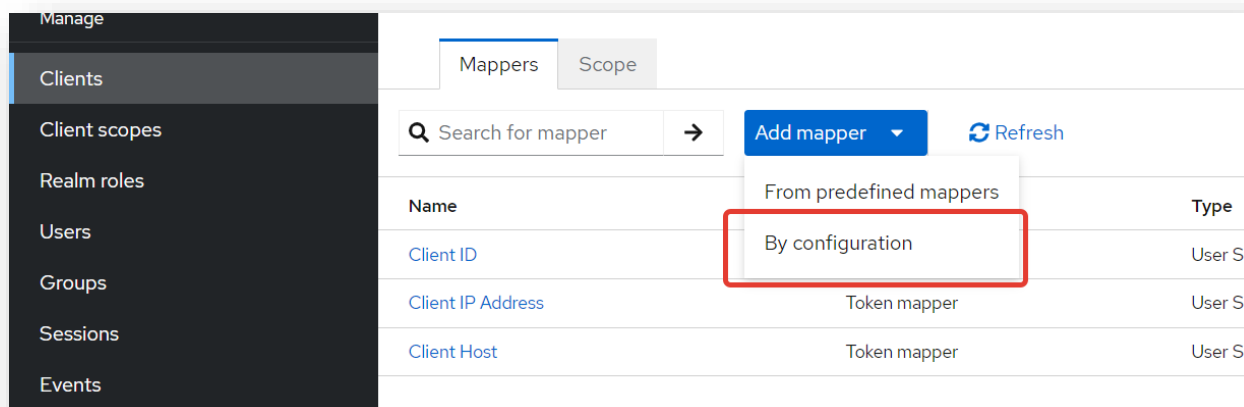
В списке клиентов выберите созданного клиента. На вкладке «Service Account roles» назначьте роль **manage-users**, чтобы приложение могло регистрировать новых пользователей



4.3.3. Передача атрибута «Avatar» в JWT токене

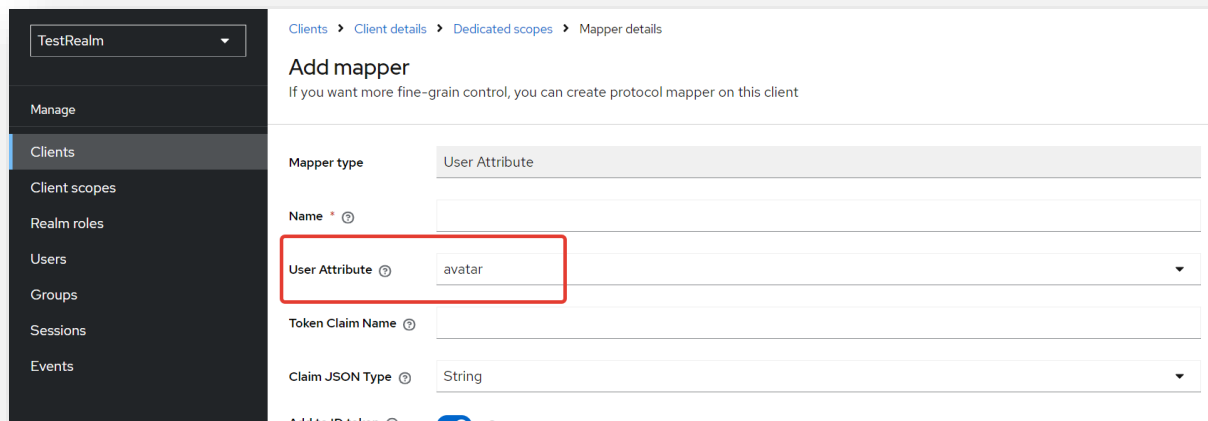
На вкладке ClientScopes выберите xxx-dedicated, где XXX – имя вашего клиента.

Выберите Add mapper -> By configuration



Выберите в списке User Attribute

Выберите атрибут avatar



TestRealm

Manage

Clients

Client scopes

Realm roles

Users

Groups

Sessions

Events

Clients > Client details > Dedicated scopes > Mapper details

Add mapper

If you want more fine-grain control, you can create protocol mapper on this client

Mapper type: User Attribute

Name * ⑤

User Attribute ⑤: avatar

Token Claim Name ⑤

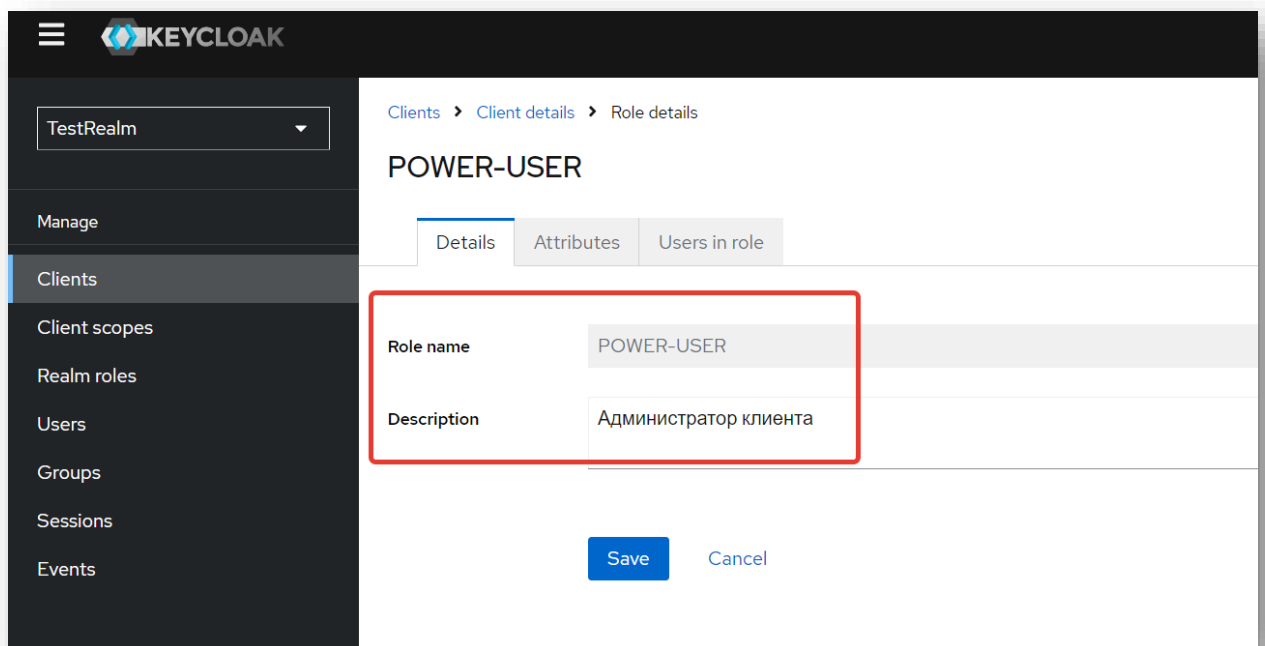
Claim JSON Type ⑤: String

Add to ID token ⑤

Теперь поле avatar будет передаваться вместе с JWT-токеном

4.3.4. Создание роли администратора

На вкладке Roles добавьте роль POWER-USER для администратора вашего приложения:



KEYCLOAK

TestRealm

Manage

Clients

Client scopes

Realm roles

Users

Groups

Sessions

Events

Clients > Client details > Role details

POWER-USER

Details Attributes Users in role

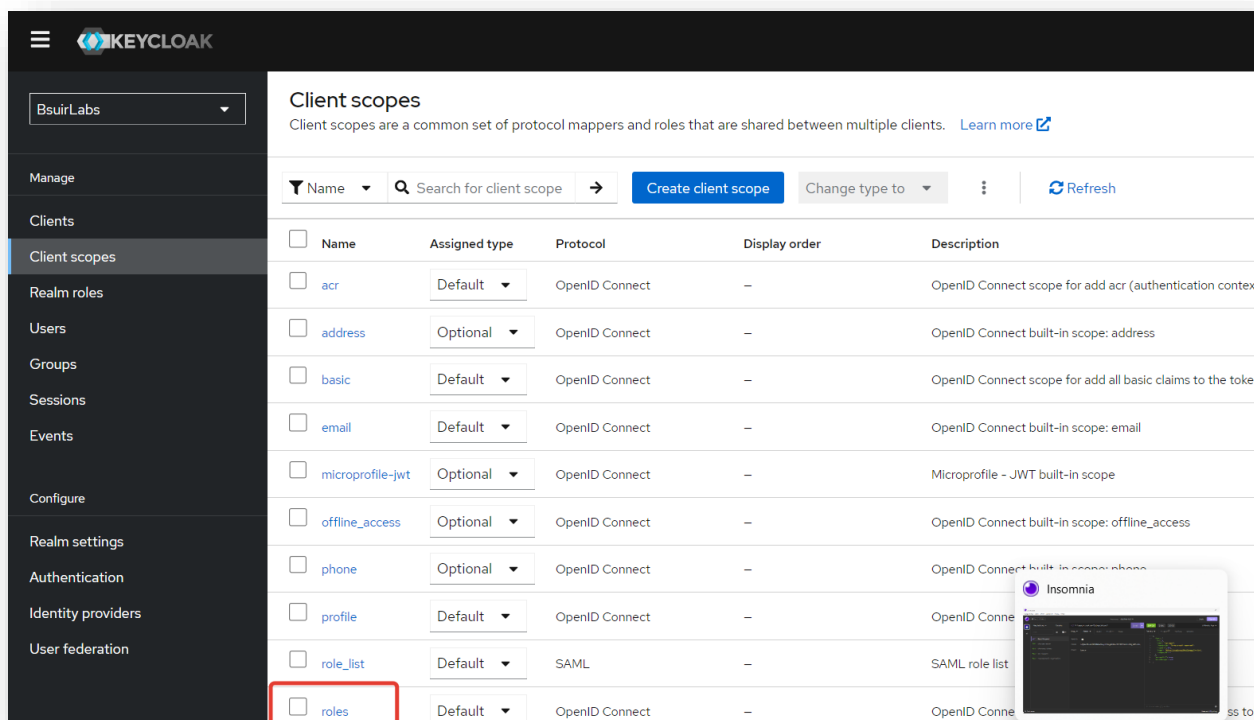
Role name: POWER-USER

Description: Администратор клиента

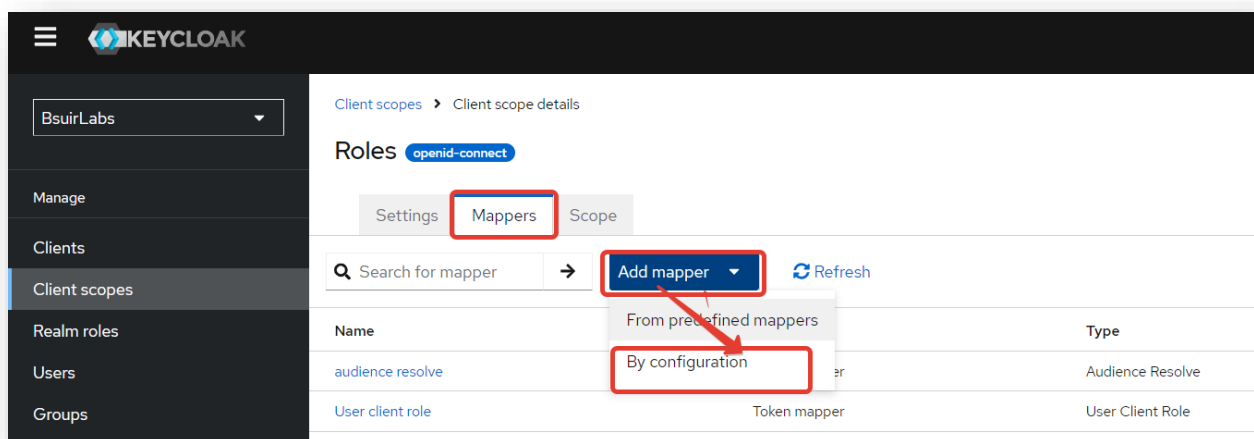
Save Cancel

4.3.5. Передача роли в токене в виде Claim

На вкладке «Client scopes» выберите «roles»

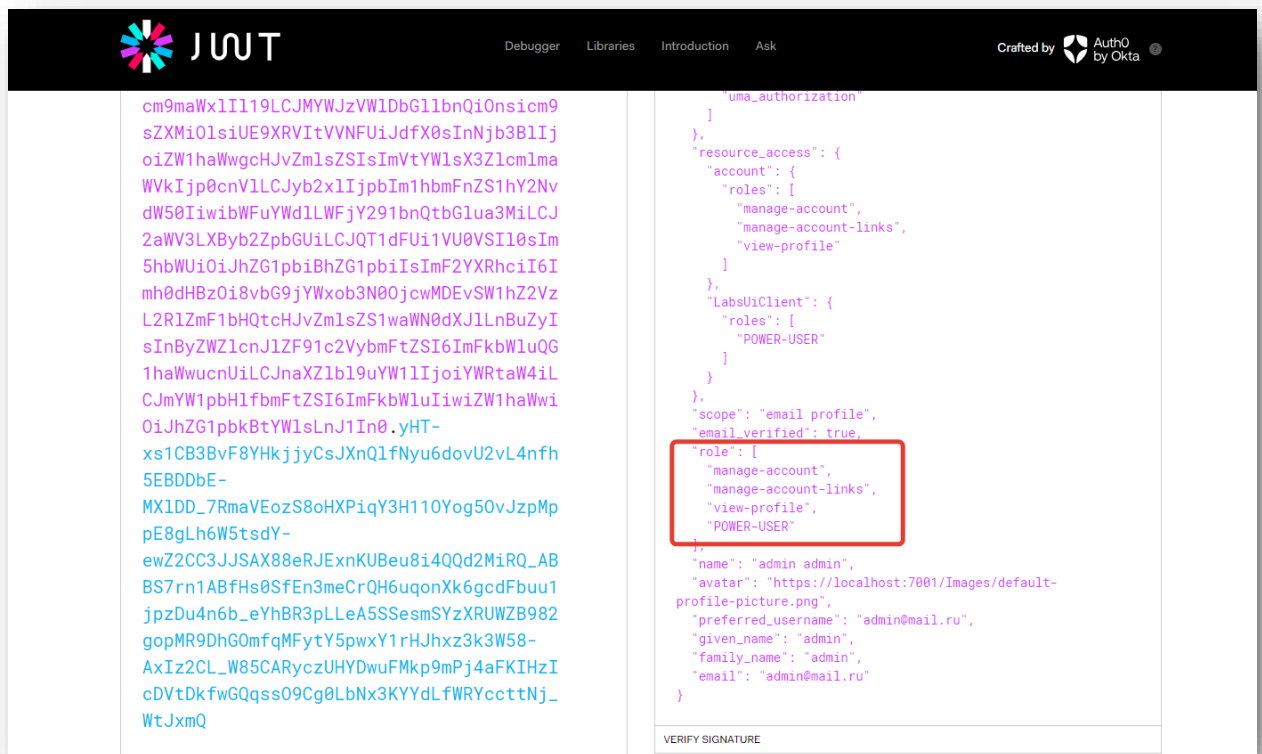


На вкладке «Mappers» добавьте Mapper «By configuration». Выберите из списка: «User client role».



- «Multivated» должно быть «On»
- «Token claim name» должно быть «role»
- «Add to access token» должно быть «On»

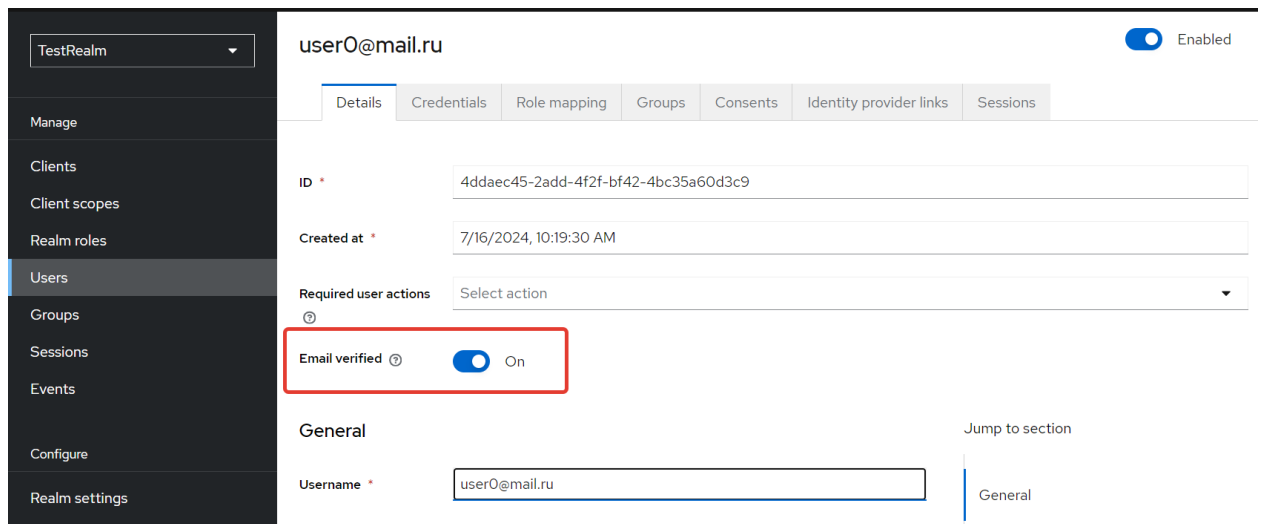
Теперь роли пользователя будут передаваться вместе с токеном в виде Claim.



4.3.6. Создание пользователей

На вкладке «Users» создайте пользователя

Укажите, что электронный адрес подтвержден



На вкладке Credentials укажите пароль пользователя. Укажите, что пароль не временный.

Создайте *еще одного* пользователя — администратора вашего приложения. Выберите созданного пользователя в списке пользователей. На

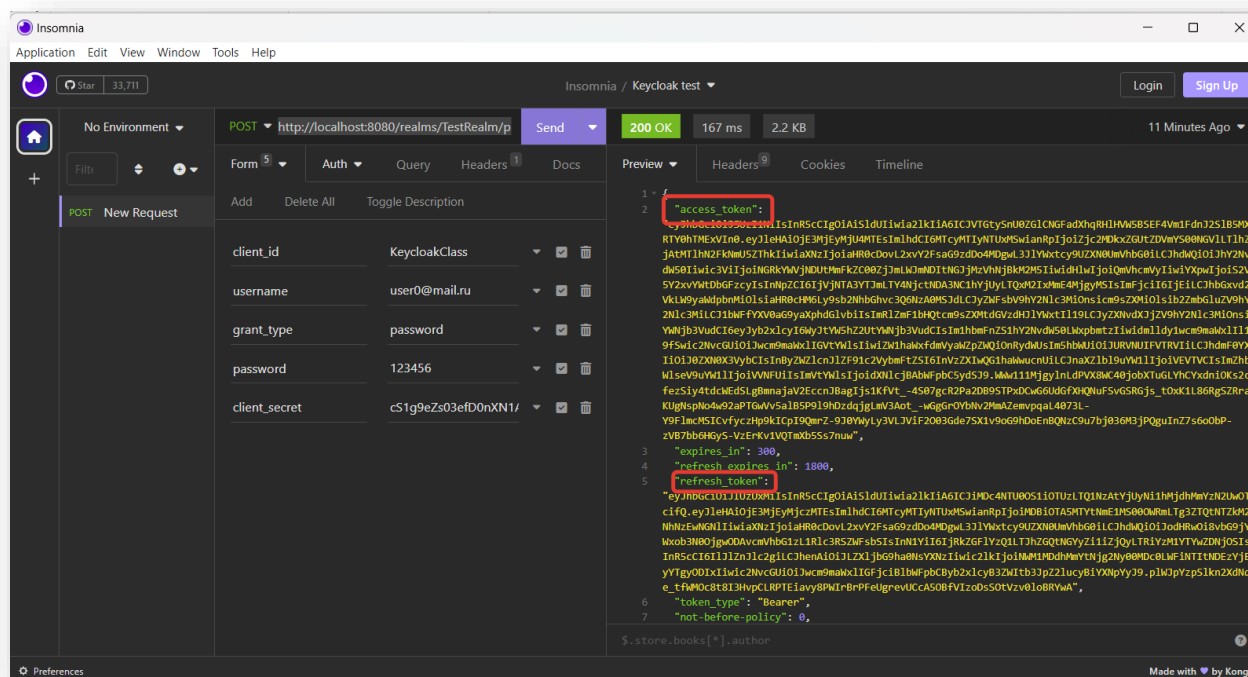
Используйте **Postman** или **Insomnia**.

Выполните запрос по методу Post к конечной точке **token endpoint**

В теле формы укажите:

- `client_id` – Id созданного клиента
- `u`
- `grant_type` – password
- `password` – пароль созданного пользователя
- `client_secret` – client secret для созданного клиента

Ваше электронный адрес создателя пользователя:



Измените тело формы:

- `client_id` – Id созданного клиента
- `grant_type` – `client_credentials`
- `client_secret` – `client secret` для созданного клиента

Отправьте запрос. В ответ должен быть получен JWT-токен для

Клиента

4.5. Настройка проекта XXX.API

Добавьте в проект XXX.Api пакет NuGet

Microsoft.AspNetCore.Authentication.JwtBearer

В файле appsettings.json добавьте секцию с данными сервера аутентификации:

```
"AuthServer": {
  "Host": "http://localhost:8080",
  "Realm": "[Имя вашего Realm]"
}
```

В папке Models создайте класс, описывающий данные сервера аутентификации:

```
internal class AuthServerData
{
    public string Host { get; set; }
    public string Realm { get; set; }
}
```

В классе Program зарегистрируйте сервис аутентификации:

```
var authServer = builder.Configuration
    .GetSection("AuthServer")
    .Get<AuthServerData>();

// Добавить сервис аутентификации
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, o =>
    {
        // Адрес метаданных конфигурации OpenID
        o.MetadataAddress = $"{authServer.Host}/realms/{authServer.Realm}/.well-known/openid-configuration";

        // Authority сервера аутентификации
        o.Authority = $"{authServer.Host}/realms/{authServer.Realm}";

        // Audience для токена JWT
        o.Audience = "account";

        // Запретить HTTPS для использования локальной версии Keycloak
        // В рабочем проекте должно быть true
        o.RequireHttpsMetadata = false;
    });
```

Опишите политику авторизации «admin», требующую наличие Claim с именем «role» и значением «POWER-USER».

```
builder.Services.AddAuthorization(opt =>
{
    opt.AddPolicy("admin", p => p.RequireRole("POWER-USER"));
});
```

В классе Program добавьте использование middleware аутентификации.

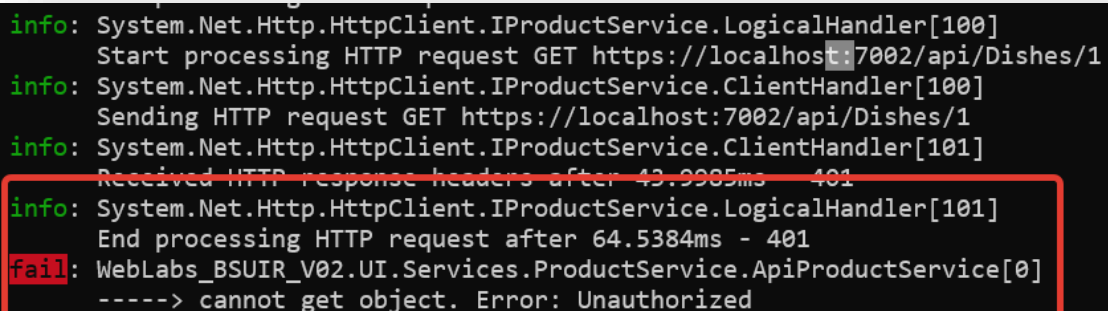
```
app.UseAuthentication();
app.UseAuthorization();
```

В контроллере API для объектов (не категорий) разрешите неавторизованный доступ только на чтение данных. Для остальных запросов используйте созданную политику «admin»

4.5.1. Проверка авторизации API

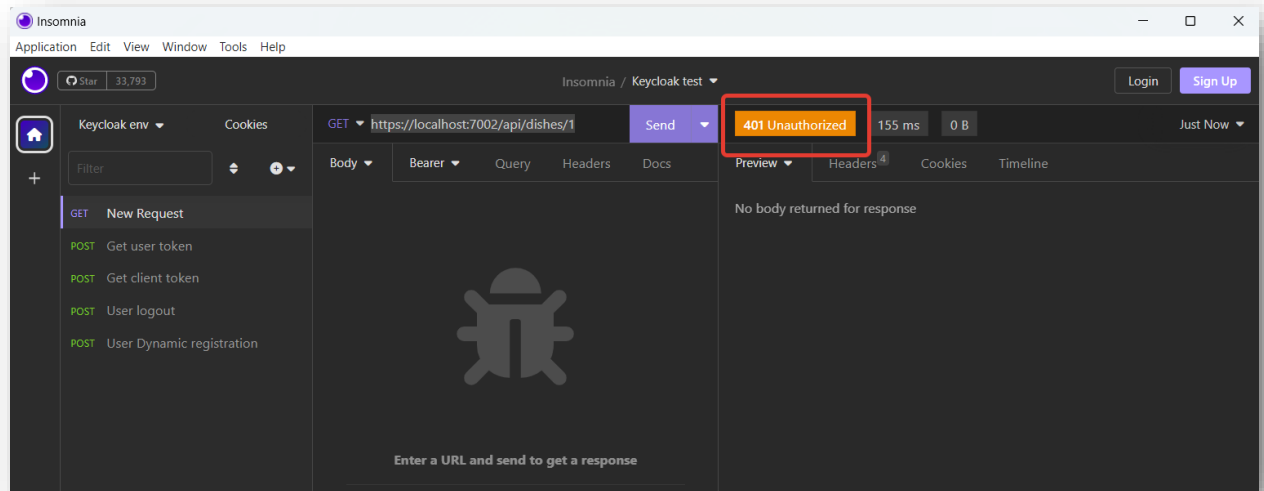
Запустите проект. Убедитесь, что в режиме администратора не выполняются действия, связанные с изменением данных.

В консоли приложения найдите запись о получении кода 401 (Unauthorized)

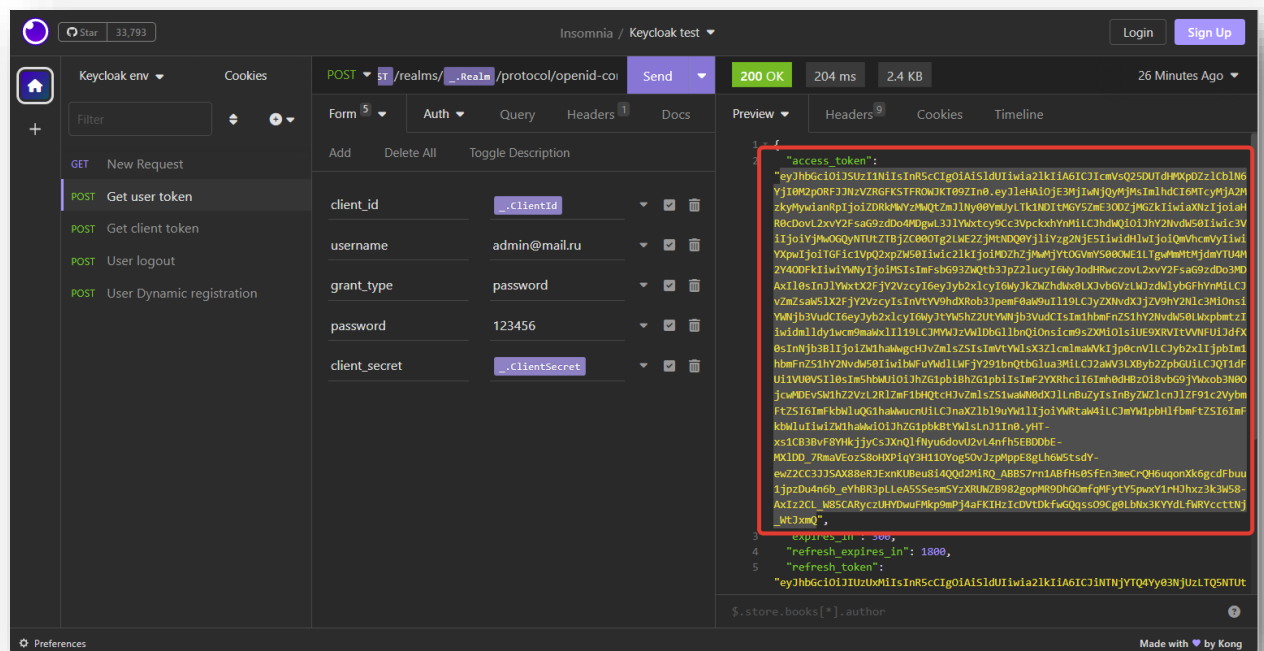


```
info: System.Net.Http.HttpClient.IProductService.LogicalHandler[100]
      Start processing HTTP request GET https://localhost:7002/api/Dishes/1
info: System.Net.Http.HttpClient.IProductService.ClientHandler[100]
      Sending HTTP request GET https://localhost:7002/api/Dishes/1
info: System.Net.Http.HttpClient.IProductService.ClientHandler[101]
      Received HTTP response headers after 43.9985ms - 401
info: System.Net.Http.HttpClient.IProductService.LogicalHandler[101]
      End processing HTTP request after 64.5384ms - 401
fail: WebLabs_BSUIR_V02.UI.Services.ProductService.ApiProductService[0]
      ----> cannot get object. Error: Unauthorized
```

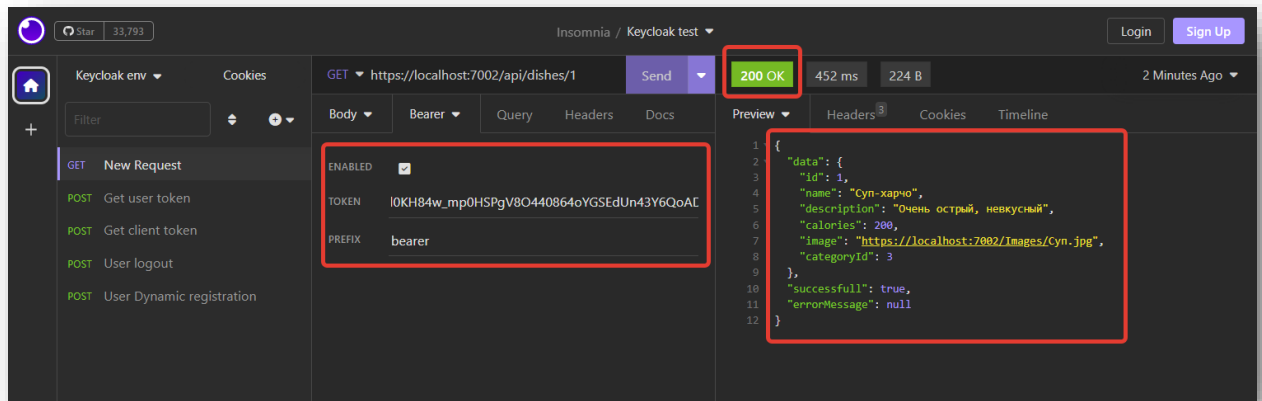
В Postman или Insomnia выполните запрос к API для получения одного объекта по ID (пример запроса: <https://localhost:7002/api/dishes/1>). Должен быть получен код 401 (Unauthorized)



Отправьте запрос для получения токена пользователя, у которого назначена роль POWER-USER. Скопируйте «access-token».



В запросе к API для получения одного объекта по ID добавьте заголовок авторизации типа «bearer», вставьте скопированный access-token. Отправьте запрос. Должен быть получен запрошенный объект:



4.6. Настройка проекта XXX.UI

Добавьте в проект пакеты NuGet
[Microsoft.AspNetCore.Authentication.OpenIdConnect](#)
[Microsoft.AspNetCore.Authentication.JwtBearer](#)

В файле appsettings.json добавьте секцию с данными для подключения к Keycloak:

```
"Keycloak": {
  "Host": "http://localhost:8080",
  "Realm": [Имя вашего Realm],
  "ClientId": [Id вашего клиента],
  "ClientSecret": [Client secret вашего клиента]
}
```

В папке HelperClasses опишите класс, инкапсулирующий данные для подключения к Keycloak:

```
internal class KeycloakData
{
    public string Host { get; set; }
    public string Realm { get; set; }
    public string ClientId { get; set; }
    public string ClientSecret { get; set; }
}
```

В классе HostingExtensions зарегистрируйте конфигурацию Keycloak:

```
builder.Services
    .Configure<KeycloakData>(builder.Configuration.GetSection("Keycloak"));
```


В классе Program добавьте аутентификацию Cookie, OpenIdConnect и JwtBearer.

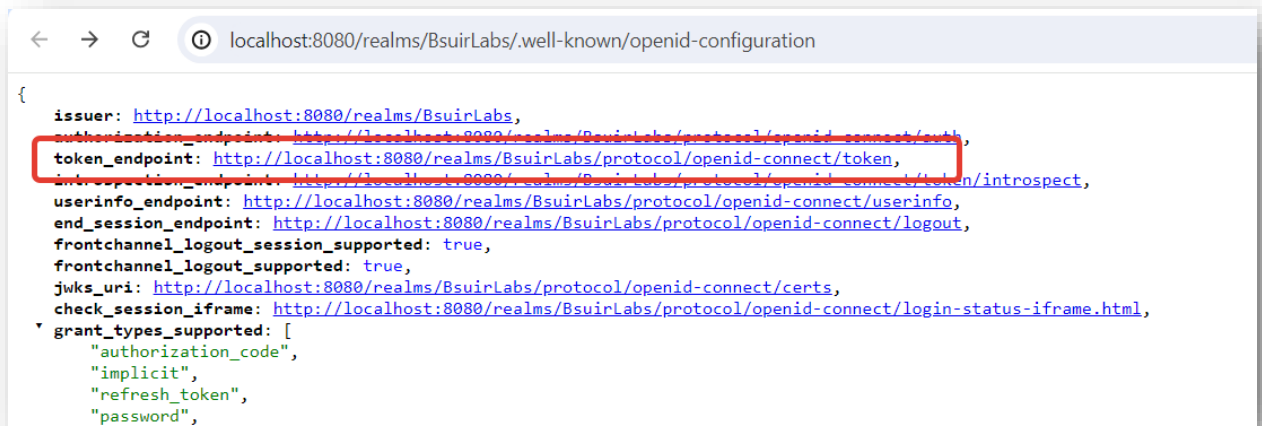
```
var keycloakData =
builder.Configuration.GetSection("Keycloak").Get<KeycloakData>();

builder.Services
    .AddAuthentication(options =>
    {
        options.DefaultScheme =
CookieAuthenticationDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme =
OpenIdConnectDefaults.AuthenticationScheme;
    })
    .AddCookie()
    .AddJwtBearer()
    .AddOpenIdConnect(options =>
    {
        options.Authority =
"${keycloakData.Host}/auth/realms/{keycloakData.Realm}";
        options.ClientId = keycloakData.ClientId;
        options.ClientSecret = keycloakData.ClientSecret;
        options.ResponseType = OpenIdConnectResponseType.Code;
        options.Scope.Add("openid"); // Customize scopes as needed
        options.SaveTokens = true;
        options.RequireHttpsMetadata = false; // позволяет обращаться к
локальному Keycloak по http
        options.MetadataAddress =
"${keycloakData.Host}/realms/{keycloakData.Realm}/.well-known/openid-
configuration";
    });
```

4.6.1. Получение токена аутентификации

Для того, чтобы получить доступ к API, необходимо передать токен аутентификации (access-token). Токен получается приложением либо при входе в систему через Keycloak, либо при обращении к конечной точке токена на сервере Keycloak (см. п. 4.4.).

Конечную точку получения токена можно найти на странице свойств Realm. Пример конечной точки:



Для получения токена создайте сервис. В папку Services добавьте папку Authentication.

Опишите интерфейс ITokenAccessor:

```

public interface ITokenAccessor
{
    /// <summary>
    /// Получение access-token
    /// </summary>
    /// <returns></returns>
    Task<string> GetAccessTokenAsync();
    /// <summary>
    /// Добавление заголовка Authorization : bearer
    /// </summary>
    /// <param name="httpClient">HttpClient, в который добавляется
    заголовок</param>
    /// <returns></returns>
    Task SetAuthorizationHeaderAsync(HttpClient httpClient);
}

```

Опишите класс KeycloakTokenAccessor, реализующий интерфейс ITokenAccessor.

При реализации метода GetTokenAsync нужно учитывать следующее:

Если пользователь вошел в систему, то токен можно извлечь из объекта HttpContext. Для этого нужно:

- в классе Program зарегистрировать сервис **HttpContextAccessor**:
`builder.Services.AddHttpContextAccessor();`
- в конструкторе класса KeycloakAuthService получить контекст:

```
_httpContext = httpContextAccessor.HttpContext;
```

Теперь можно получить токен:

```
var token = await _httpContext.GetTokenAsync("access_token");
```

Если пользователь не входил в систему, то нужно получить токен клиента. Для этого в конструктор класса KeycloakAuthService нужно внедрить данные сервера Keycloak (зарегистрированы как IOptions<KeycloakData>) и HttpClient.

Зарегистрируйте созданный сервис в классе HostingExtensions:

```
builder.Services.AddHttpClient<ITokenAccessor, KeycloakTokenAccessor>();
```

Пример реализации:

```
public class KeycloakTokenAccessor : ITokenAccessor
{
    private readonly KeycloakData _keycloakData;
    private readonly HttpContext? _httpContext;
    private readonly HttpClient _httpClient;

    public KeycloakTokenAccessor(IOptions<KeycloakData> options,
                                IHttpContextAccessor httpContextAccessor,
                                HttpClient httpClient)
    {
        _keycloakData = options.Value;
        _httpContext = httpContextAccessor.HttpContext;
        _httpClient = httpClient;
    }

    public async Task<string> GetAccessTokenAsync()
    {
        // Если пользователь вошел в систему, получить его токен
        if (_httpContext?.User?.Identity?.IsAuthenticated)
        {
            return await _httpContext.GetTokenAsync("access_token");
        }

        // Если пользователь не входил в систему, получить токен клиента

        // Keycloak token endpoint
        var requestUri =
            $"{_keycloakData.Host}/realms/{_keycloakData.Realm}/protocol/openid-
            connect/token";

        // Http request content
        HttpContent content = new FormUrlEncodedContent([
            new KeyValuePair<string, string>("client_id", _keycloakData.ClientId),
            new KeyValuePair<string, string>("grant_type", "client_credentials"),
```

```

        new
KeyValuePair<string, string>("client_secret", _keycloakData.ClientSecret)
    });

    // send request
    var response = await _httpClient.PostAsync(requestUri, content);

    if (!response.IsSuccessStatusCode)
    {
        throw new HttpRequestException(response.StatusCode.ToString());
    }

    // extract access token from response
    var jsonString = await response.Content.ReadAsStringAsync();
    return JsonObject.Parse(jsonString)["access_token"].GetValue<string>();
}

public async Task SetAuthorizationHeaderAsync(HttpClient httpClient)
{
    string token = await GetAccessTokenAsync();

    httpClient
        .DefaultRequestHeaders
        .Authorization = new AuthenticationHeaderValue("bearer", token); ;
}
}

```

4.6.2. Использование токена в сервисе ApiProductService

Внедрите сервис ITokenAccessor в конструктор класса ApiProductService.

Во всех методах класса ApiProductService перед отправкой запроса http вызывайте метод `SetAuthorizationHeaderAsync` внедренного сервиса.

4.6.3. Использование токена в классе FileService

Внедрите сервис ITokenAccessor в конструктор класса FileService.

Перед отправкой запроса на сохранения файла вызывайте метод `SetAuthorizationHeaderAsync` внедренного сервиса.

4.6.4. Проверка работы

Запустите проект. Убедитесь, что в режиме администратора данные модифицируются.

4.7. Регистрация на сервере аутентификации

Задача: создать страницу регистрации нового пользователя.

- Предусмотреть сохранение аватара пользователя.
- Аватар сохранять в папке wwwroot/Images проекта XXX.API.
- Имя файла аватара назначать случайным образом.
- Url файла аватара сохранять в атрибуте «Avatar» пользователя.
- Если аватар отсутствует, в атрибут «Avatar» записать Url стандартной картинки анонимного пользователя.

4.7.1. Общая информация

Сервер Keycloak предоставляет страницу регистрации. Но ее функционал не предусматривает сохранение файла изображения.

Создадим свою страницу регистрации, а для сохранения данных нового пользователя воспользуемся Admin API сервера Keycloak (см. https://www.keycloak.org/docs-api/latest/rest-api/index.html#_users)

4.7.2. Интерфейс сервиса аутентификации

В папке Authorization проекта XXX.UI опишите интерфейс IAuthService, предоставляющий метод для регистрации нового пользователя:

```
public interface IAuthService
{
    /// <summary>
    /// Регистрация пользователя на сервере аутентификации
    /// </summary>
    /// <param name="email">email нового пользователя</param>
    /// <param name="password">пароль нового пользователя</param>
    /// <param name="avatar">объект файла аватара пользователя</param>
    /// <returns>Result - признак успешного добавления пользователя
    /// ErrorMessage - сообщение об ошибке</returns>
    Task<(bool Result, string ErrorMessage)> RegisterUserAsync(string email,
                                                                string password,
                                                                IFormFile? avatar);
}
```

4.7.3. Создание контроллера Account

Для регистрации пользователя в папке Models опишите класс, инкапсулирующий данные, необходимые для регистрации нового пользователя:

```
internal class RegisterUserViewModel
{
    [Required]
    public string Email { get; set; }
    [Required]
    public string Password { get; set; }
    [Required]
    [Compare(nameof(Password))]
    public string ConfirmPassword { get; set; }
    public IFormFile? Avatar { get; set; }
}
```

Добавьте в проект пустой контроллер MVC с именем AccountController.

Опишите методы (Action) для регистрации нового пользователя. В качестве модели представления используйте класс RegisterUserViewModel. Требуется два метода. Первый – по методу Get отправляет клиенту форму для ввода регистрационных данных. Второй – по методу Post принимает данные и регистрирует пользователя на сервере аутентификации.

Используйте метод RegisterUserAsync объекта IAuthService. Для этого внедрите этот объект в метод Register:

```
public IActionResult Register()
{
    return View(new RegisterUserViewModel());
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Register(RegisterUserViewModel user,
    [FromServices] IAuthService authService)
{
    if (ModelState.IsValid)
    {
        if (user == null)
        {
            return BadRequest();
        }
    }
}
```

```

        var result = await authService.RegisterUserAsync(user.Email,
                                                         user.Password,
                                                         user.Avatar);

        if (result.Result)
        {
            return Redirect(Url.Action("Index", "Home"));
        }
        else return BadRequest(result.ErrorMessage);
    }
    return View(user);
}

```

4.7.4. Представление Register

Создайте представление для Action Register контроллера Account. Воспользуйтесь scaffold (шаблонный элемент). Используйте шаблон Create. В качестве модели укажите класс `RegisterUserViewModel`.

Откройте созданное представление.

Добавьте в форму атрибут `enctype="multipart/form-data"`

Добавьте разметку для загрузки файла аватара

4.7.5. Реализация метода RegisterUserAsync интерфейса IAuthService

В папке Authorization опишите класс KeycloakAuthService, реализующий интерфейс IAuthService.

Для регистрации пользователя в Keycloak необходимо выполнить следующие шаги:

- 1) Получить JWT access-token-токен клиента.
- 2) Послать запрос на API администратора (https://www.keycloak.org/docs-api/latest/rest-api/index.html#_users). Конечная точка для регистрации пользователя: POST /admin/realms/{realm}/users.

В запросе установить заголовок Authorization: bearer { access-token }

С запросом передать StringContent, который содержит данные о новом пользователе в формате JSON (<https://www.keycloak.org/docs-api/latest/rest-api/index.html#UserRepresentation>). Пример данных о новом пользователе:

```

var userData = ""
{
    "attributes": {

```

```

        "avatar" : "images/default-profile-picture.png"
    },
    "username": "igor@example.com",
    "email": "igor@example.com",
    "enabled": true,
    "emailVerified":true,
    "credentials": [{
        "temporary":false,
        "type": "password",
        "value": "123456"
    }]
}
""";

```

Данные о новом пользователе можно сформировать, используя механизм интерполяции строк C#.

В приведенном ниже примере будет использоваться другой подход - сериализация объектов класса в JSON. Для этого опишите классы, инкапсулирующие данные о пользователе и данные Credentials. Поскольку эти классы не будут нигде больше использоваться, можно описать их в файле класса `KeycloakAuthService`:

```

class CreateUserModel
{
    public Dictionary<string, string> Attributes { get; set; } = new();
    public string Username { get; set; }
    public string Email { get; set; }
    public bool Enabled { get; set; } = true;
    public bool EmailVerified { get; set; } = true;
    public List<UserCredentials> Credentials { get; set; } = new();
}

class UserCredentials
{
    public string Type { get; set; } = "password";
    public bool Temporary { get; set; } = false;
    public string Value { get; set; }
}

```

Для отправки запросов к Keycloak внедрите в конструктор класса `KeycloakAuthService` объекты `HttpClient` и `IOptions<KeycloakData>`.

Для сохранения файла аватара внедрите `IFileService`.

Пример реализации:

```

public class KeycloakAuthService : IAuthService
{
    private readonly HttpClient _httpClient;
    private readonly IFileService _fileService;
    private readonly ITokenAccessor _tokenAccessor;
    KeycloakData _keycloakData;

    public KeycloakAuthService(HttpClient httpClient,
        IOptions<KeycloakData> options,
        IFileService fileService,
        ITokenAccessor tokenAccessor)
    {
        _httpClient = httpClient;
        _fileService = fileService;
        _tokenAccessor = tokenAccessor;
        _keycloakData = options.Value;
    }

    public async Task<(bool Result, string ErrorMessage)> RegisterUserAsync(
        string email,
        string password,
        IFormFile? avatar)
    {
        // добавить JWT token в заголовки
        try
        {
            await _tokenAccessor.SetAuthorizationHeaderAsync(_httpClient);
        }
        catch (Exception ex)
        {
            return (false, ex.Message);
        }

        var avatarUrl = "/images/default-profile-picture.png";
        // сохранить Avatar, если аватар был передан при регистрации
        if (avatar != null)
        {
            var result = await _fileService.SaveFileAsync(avatar);
            if (result != null) avatarUrl = result;
        }

        // Подготовка данных нового пользователя
        var newUser = new CreateUserModel();
        newUser.Attributes.Add("avatar", avatarUrl);
        newUser.Email = email;
        newUser.Username = email;
        newUser.Credentials.Add(new UserCredentials { Value = password });

        // Keycloak user endpoint
    }
}

```

```

        var requestUri =
$"{_keycloakData.Host}/admin/realms/{_keycloakData.Realm}/users";

        // Подготовить контент запроса
        var serializerOptions = new JsonSerializerOptions
        {
            PropertyNamingPolicy = JsonNamingPolicy.CamelCase
        };
        var userData = JsonSerializer.Serialize(newUser, serializerOptions);
        HttpContent content = new StringContent(userData, Encoding.UTF8,
"application/json");

        // Отправить запрос
        var response = await _httpClient.PostAsync(requestUri, content);

        if (response.IsSuccessStatusCode) return (true, String.Empty);
        return (false, response.StatusCode.ToString());
    }
}

```

4.7.6. Проверка страницы регистрации

Запустите проект, введите адрес <https://localhost:7002/account/register>

Зарегистрируйте нового пользователя:

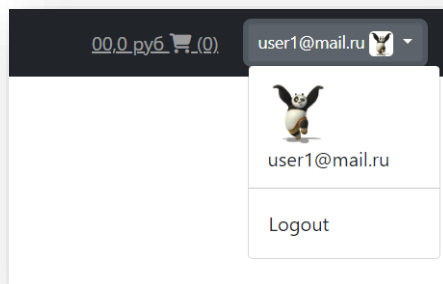
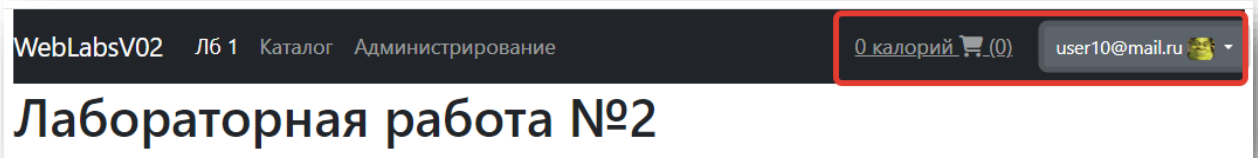
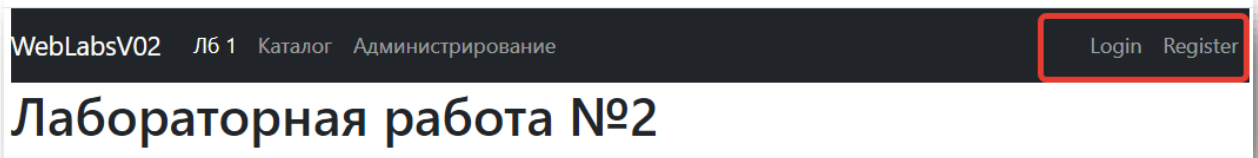
4.8. Доработка проекта XXX.UI

4.8.1. Задание

В основном проекте требуется реализовать следующее: информация, выводимая в меню «Информация пользователя», должна зависеть от того, вошел пользователь в систему, или нет. А именно:

- если пользователь не вошел в систему, показывать кнопки «Войти» («Login») и «Зарегистрироваться» («Register»).

- если пользователь вошел в систему, то показывать корзину, имя пользователя и аватар. Кнопка «Logout» должна работать.



4.8.2. Рекомендации к заданию

а) для реализации функций Login и Logout добавьте в контроллер Account два действия (action): Login, реализующее переадресацию на страницу входа сервера аутентификации, и Logout (по методу HttpPost), реализующий выход. Пример реализации:

```
public async Task Login()
{
    await HttpContext.ChallengeAsync(
        OpenIdConnectDefaults.AuthenticationScheme,
        new AuthenticationProperties { RedirectUri = Url.Action("Index",
            "Home") });
}
[HttpPost]
public async Task Logout()
{
    await
    HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
    await HttpContext.SignOutAsync(OpenIdConnectDefaults.AuthenticationScheme,
```

```

        new AuthenticationProperties { RedirectUri = Url.Action("Index",
            "Home") });
    }

```

b) В частичном представлении `_UserInfoPartial`:

Для определения того, что пользователь аутентифицирован, можно использовать свойство `User`:

`User.Identity.IsAuthenticated`

Имя пользователя можно получить из соответствующего утверждения (Claim). Аватар можно получить из соответствующего утверждения (Claim):

```

@{
    var name = @User
        .Claims
        .FirstOrDefault(c => c.Type.Equals("preferred_username",
            StringComparison.OrdinalIgnoreCase))?
        .Value;

    var avatar = @User
        .Claims
        .FirstOrDefault(c => c.Type.Equals("avatar",
            StringComparison.OrdinalIgnoreCase))?
        .Value;
}

```

c) Кнопки Login и Logout должны адресовать на соответствующие методы контроллера Identity – см. п. а) (использовать тэг-хелперы)

Запустите проект. Проверьте возможность входа/выхода в систему, а также правильность работы меню «Информация пользователя».

5. Контрольные вопросы

1. Какой механизм аутентификации имеет встроенную поддержку в ASP.Net Core?
2. Что описывают классы ClaimsPrincipal и ClaimsIdentity?
3. Как подключить Middleware аутентификации и авторизации?
4. Приведите пример использования свойства HttpContext.User.
5. Как в коде проверить, что пользователь прошел аутентификацию?
6. Как получить значение Claim пользователя?
7. Как получить Id пользователя, прошедшего аутентификацию?
8. Как разрешить доступ к контроллеру только для пользователей с ролью «manager»?
9. Как создать политику авторизации с помощью Claim?
10. Как создать куки аутентификации с помощью объекта HttpContext?
11. Как добавить в проект использование системы членства Microsoft.AspNetCore.Identity?
12. Как с помощью системы членства Microsoft.AspNetCore.Identity создать нового пользователя?
13. Как с помощью системы членства Microsoft.AspNetCore.Identity осуществить вход пользователя в систему?
14. Как с помощью системы членства Microsoft.AspNetCore.Identity добавить Claim пользователю?
15. Какой интерфейс используется в Microsoft.AspNetCore.Identity для доступа к хранилищу пользователей?